An Improved SMT-Based Scheduling for Overloaded Real-Time Systems

Shimin Wang, Xiaojuan Liao, Min Wang, Luyue Chang, Huan Yang, and Tao Wang

Abstract—In real-time systems, completing tasks by their deadlines is as significant as getting the correct results for tasks. Once the workload exceeds the system capacity, it is difficult to complete all tasks correctly before the deadlines. As a result, some tasks will miss the deadlines, leading to system performance degradation or security issues. To alleviate system performance degradation caused by overload, Cheng et al. proposed an SMT-based scheduling algorithm, finding that a SMT solver is an efficient tool to solve the scheduling problem in overloaded real-time systems. However, we notice that their method has a large amount of redundant encoding. Moreover, the SMT solver must be called in a loop to calculate the optimal solution, which increases the computational cost and reduces the efficiency. This paper improves the method proposed by Cheng et al. by eliminating redundant encoding and successive calls of the SMT solver, thus making the SMT-based algorithm more compact and productive. The experimental results show that the updated SMT-based method can significantly improve the system performance.

Index Terms—optimal scheduling, SMT, overloaded system, real-time system.

I. Introduction

A. Background

CONCURRENCY is one of the most important features of computer systems. Two program fragments cannot be executed simultaneously on a uniprocessor computer, and the execution orders are different. An efficient task scheduling algorithm is needed to guarantee the concurrent computing of a computer system. In the past several decades, real-time systems are widespread in daily life and industrial production, and play an increasingly important role in nowaday applications. For instance, intrusion detection systems [1], anti-counterfeiting systems [2], and flight control systems are all commonly used in real-time systems.

In hard real-time systems, all tasks of a task set should be completed correctly by the deadlines so as to prevent overload [3]. Under normal workload, classical scheduling algorithms ensure that all tasks can be completed before their deadlines. However, in practical applications, once the workload exceeds the system capacity, scheduling algorithm can hardly complete all tasks by their deadlines. This is called system overload. For the sake of alleviating the system performance degradation, an efficient algorithm suitable for overloaded real-time systems is desirable.

So far, many scheduling algorithms have been proposed to optimally solve the scheduling problem of overloaded real-time systems. Various objectives of real-time systems can be considered [4], [5]. On the one hand, when a missed deadline corresponds to a disgruntled customer, the goal is to satisfy as many customers as possible [6]. This leads to the goal of maximizing the number of completed tasks. On the other hand, if a missed deadline corresponds to a missed first prize in the lottery, the goal is to get as many first prizes as possible to maximize the total money awards rather than the number of prizes. In this case, the goal is to maximize the total weights of the completed tasks. In this paper, both scheduling goals are implemented.

B. Related works

Considering the high computational complexity of the overload problem, previous scheduling algorithms mainly focus on near optimal solutions. For example, Tres et al. [7] proposed a dynamic miss based (DMB) algorithm, which can change task values to adjust their importance according to the timing faults rate. Marchand et al. [8] designed schemes for a real-time system with skippable tasks, whose deadlines are likely to be missed. In their work, each task is assigned to a skip parameter, which shows the tolerance of the task to miss the deadline with the change of practical environment. Cheng et al. proposed dynamic programming with congestion control mechanism (DPSC) [9], and they put forward Greedy scheduling with feedback control (GSFC) subsequently [10]. Based on the optimal scheduling results of the currently known tasks, DPSC and GSFC can partially deal with uncertain new tasks. Althouth the dynamic scheduling algorithms can seek approximate solutions, they always fail to make the result steadily approach to the optimal solution.

Another research line is devoted to designing static sheduling algorithms. Compared with the dynamic scheduling algorithms, they can determine the execution order of the tasks in advance through static analysis at compiling time, so that potential risks may be predicted at the early stage of design. Typical static sheduling algorithms include constraint programming, which can tackle the task scheduling problem as a natural generalization [11], and SAT (Boolean Satisfiability) formulation [12], [13] that is based on Boolean variables and logical operations. Nowadays, many satisfiability problems have been solved by SAT solvers [14], including the polynomial

Manuscript received May 26, 2019. This work was supported in part by National Natural Science Foundation of China under Grant 61806171, Ministry of Education in China Project of Humanities and Social Sciences under Grant 17YJCZH260.

Corresponding authors S. Wang (e-mail: cdut_wsm@163.com), X. Liao (e-mail: liao_xiaojuan@126.com) and M. Wang (e-mail: min_wang126@126.com) are with the School of Cyberspace Security, Chengdu University of Technology, Chengdu City, Sichuan Province, China.

size asymmetric linear model proposed by Sergey Gubin [15]. Although SAT solvers can solve large-scale satisfiability problems by their own advanced techniques and produce effective solutions within a reasonable time, they are not expressive enough for pratical applications. Also, they can only support simple logical operations such as and, or, not. In order to tackle arithmetic constraints and make full use of SAT solvers, Satifiability Modulo Theories (SMT) [16], [17] have been developed. As the extension of SAT-solvers, SMT-solvers can efficiently deal with Boolean combinations of arithmetic predicates. Motivated by great development of SMT solvers, Cheng et al. proposed a static scheduling algorithm in [18] and extended it to multi-processors in [19] in order to maximize the number of tasks before the deadlines. Subsequently they extended the scheduling target to more circumstances in [20]. Experimental results in [20] demonstrate that different scheduling goals can be achieved easily with few modifications. As we all know, it is very difficult to tackle NP-hard problems. There exist many typical NP-hard problems, such as approximate period problem [21], typical working day planning problem of indoor service robot [22], and c-fragment longest arc-preserving common subsequence problem [23]. To our delight, SMT solvers are generally employed to tackle NP-hard problems on a certain scale (e.g., Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) [24]). Bofill et al. [25] used Yices, a SMT solver, to solve MRCPSP, finding that SMT is a competitive approach for this kind of scheduling problem. SMT solvers have been used in many fileds, including the generation of automated testing. Hiroki et al. take the unique advantages of SMT solvers to solve the automated test generation problem for object-oriented programs with multiple targets, and made the generation of automated testing more successful [26].

C. Our contribution

The SMT-based formulation in [20] is superior to previous classical scheduling algorithms in overloaded real-time systems. However, it should be noted that the SMT solvers must be invoked repeatedly to find the optimum solution. In addition, we notice that there are a large number of redundant assertions in their encoding. Repeated SMT calls and redundant encoding increase the computational cost, decreasing the solving efficiency to some extent. To make the SMT encoding more compact and efficient, this paper improves the dispatching algorithm based on Cheng et al. [20] by eliminating the code redundancy and the successive calls of a SMT solver. Experimental results demonstrate that compared with the previous works [20], our updated method manages to enhance the solving efficiency by more than two orders of magnitude. The SMT-solver we adopted is Z3 [27], which is designed by Microsoft Research to check the satisfiability of a logical formula in one or more theories with powerful functionality. The first-order logic formulas presented in this paper are all based on the syntax of Z3.

D. Organization of this paper

The rest of this paper is structured as follows. Section II introduces preliminary on SMT and Z3, and Section III describes the scheduling model. The improved method is elaborated in Section IV. Section V describe the experiments. Finally, Section VI summarizes the paper.

II. Preliminary

A. Satisfiability Modulo Theories

Satisfiability Modules Theories problem is a decision-making problem which combines the classical first-order logic theory in computer science and mathematical logic [28]. It is worth nothing that SMT provides not only high efficiency, but also an expressive language for scheduling problem. SMT can be regarded as a constraint satisfiability problem. It is a formal method of constrained programming. The first-order logical formula consists of variables, quantifiers, functions, predicate symbols and logical operators. If there exists an explanation that makes the formula F true, F is satisfiable, otherwise, F is unsatisfiable. For example, in formula F: $\exists x, y \in R$, $(x + y = 1) \wedge (x > y + 0.1)$, where R is a real number set. There is an interpretation, such as $x = 0.6$ and $y = 0.4$, that makes F true. Therefore, F is satisfiable. Considering the same example, the condition "$\exists x, y \in R$" is replaced with "$\exists x, y \in Z$" in formula F, where Z is an integer set. Since we cannot find an assignment in an integer set to satisfy the formula F, we say F is unsatisfiable.

B. Z3

To date, there have been many kinds of SMT solvers for decision making problems, some of which have the extended versions for optimization problems, such as Z3 and MathSat5 [29]. The optimizing version of Z3 is νZ [30], and MathSat5's optimizing tool contains a special version called optiMathSAT [31]. Both Z3 and MathSat5 have high performance in different application scenarios. Since Z3 is more efficient in tackling scheduling problems, like the job-shop scheduling problems (JSP) [32], this paper uses Z3 as the SMT solver, which was also adopted by Cheng et al. [20]. Z3 is Microsoft research's most advanced theorem prover and it can be used to test the theoretical satisfiability of a formula. Internally, Z3 maintains a stack of formulas and declarations provided by users. The assertions begin with the command "assert", followed by a formula with prefix expressions. The command "assert" adds a formula into the Z3-solver internal stack, and prefix expression writes the operator in front and the operands in back. For example, prefix expression "$+\ a\ b$" means "$a + b$". Prefix expression is so useful that it can complete all operations in assertions only by two simple operations: out-of-stack and in-stack, thereby improving the efficiency of Z3.

In real-time systems, a Z3 solver can generate scheduling tables that tell users the order in which tasks are scheduled to execute. In order to obtain the optimal scheduling result, we are supposed to formalize the constrains with first-order logic language. The input of Z3, as a "SMT model", is generated by the first-order

logic language. When the SMT model is imported into Z3, the final solution will be returned after a period of time. The solution explains the variables defined in the model, under which all asserted logical formulas are evaluated true. After we input a SMT model into Z3, Z3 will return a non-empty model if there is a variable assignment that satisfies the constraints, otherwise an empty model will be returned. The key of this paper is to formalize the scheduling problem as a SMT model that can be handled by an off-the-shelf Z3 solver.

III. Scheduling model

For real-time systems, the task is valuable only when it correctly ends before its deadline, otherwise it is worthless. This is named firm-deadline model [33]. We adopt the firm-deadline model for uniprocessor and assume that tasks sporadically arrive at the system. The problem definition is consistent with Cheng et al.'s work [20]. For convenience, all symbols are defined in the Table I.

A real-time system has a task set $\mathcal{T}$ that contains n independent real-time tasks to be executed, i.e., $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is a four-tuple, denoted by $\tau_i = (r_i, c_i, d_i, w_i)$, where i is the index of the task. Each entry is described as follows.

- r_i is the request time instant.
- c_i is the required execution time.
- d_i is the deadline.
- w_i is the weight.

The weight w_i reflects the importance of τ_i. In a task set $\mathcal{T}$, when w_i is larger, τ_i is more important, and higher priority should be given to the task. If all tasks have indistinguishable importance, the weights of all tasks are set as any equal number. In our work, the following two situations are considered.

- Tasks with identical weights indicate that the priorities of all tasks are the same.
- Tasks with different weights indicate that tasks with higher weight values should have higher priority.

To allow preemption, each task $\tau_i \in \mathcal{T}$, is regarded as a series of indivisible fragments of f_j^i, where $\tau_i = (f_a^i, f_b^i, \ldots, f_{q_i}^i)$, q_i is the total number of fragments of τ_i, and f_a^i denotes the a^{th} fragment of the task τ_i. The symbol s_a^i denotes the start time of the a^{th} fragment of τ_i, and c_a^i denotes the execution time of the a^{th} fragment of the task. With the above definitions, $s_{q_i}^i$ denotes the start time of the last fragment of task τ_i, and $c_{q_i}^i$ denotes the execution time of the last fragment of task τ_i. When $s_{q_i}^i + c_{q_i}^i \leq d_i$, the task τ_i is completed successfully. For the symbol c_a^i, $c_i = \sum_{a=1}^{q_i} c_a^i (1 \leq a \leq q_i)$. For the symbol s_a^i, it is obvious that $s_a^i \geq r_i$ and $c_{a+1}^i \geq s_a^i + c_a^i$.

In a real scheduling environment, there are usually dependencies between tasks. For example, task τ_i needs the result of τ_j, and τ_i can only be executed after τ_j is completed. We define such dependency as $\tau_j \prec \tau_i$.

The purpose of this paper is to improve the scheduling algorithm proposed by Cheng et al. [20]. When overload occurs in real-time systems, we seek to obtain the optimal scheduling scheme with higher efficiency. The scheduling target taken into consideration are two-folded, which is summarized as follows:

(1) If all tasks have identical weights, the goal is to maximize the total number of tasks to be executed before their deadlines.
(2) If the tasks are associated with distinct weights, the goal is to maximize the sum of weights of the tasks to be executed before their deadlines.

TABLE I
Symbols and Explanations in this paper

Symbol	Explanation
t	System time instant
$\mathcal{T}$	A set of real-time tasks
τ_i	A real-time task $\tau_i \in \mathcal{T}$, i is the index of the task
r_i	The request time instant of τ_i
c_i	The required execution time of τ_i
d_i	The deadline of τ_i
w_i	The weight of τ_i
q_i	The number of indivisible fragment in τ_i
f_a^i	The a^{th} indivisible fragment of τ_i
s_a^i	The start execution time of f_a^i
c_a^i	The execution time of f_a^i
$\tau_j \prec \tau_i$	τ_i relies on τ_j
ES_a^i	The earliest start execution time of f_a^i
LS_a^i	The latest start execution time of f_a^i
EC_a^i	The earliest completion time of f_a^i
LC_a^i	The latest completion time of f_a^i
$\mathbb{T}_i$	The completion status of τ_i
$\mathbb{W}_i$	The weight value obtained by τ_i
$\mathbb{C}_i$	The completion flag of τ_i

To maximize the number of the completed tasks in the system, we use symbol $\mathbb{T}_i$ to represent the completion status of τ_i. $\mathbb{T}_i = 1$ if τ_i accomplishes before its deadline, otherwise $\mathbb{T}_i = 0$. To maximize the sum of weight of the completed tasks, we use symbol $\mathbb{W}_i$ to represent the weight value obtained by τ_i. If τ_i accomplishes before its deadline in the context of tasks with distinct importance, $\mathbb{W}_i$ is assigned with w_i, otherwise $\mathbb{W}_i$ equals 0. To facilitate the subsequent elaboration, we use symbol $\mathbb{C}_i$ as a completion flag to indicate whether τ_i is completed before its deadline. $\mathbb{C}_i = true$ if τ_i accomplishes before its deadline, otherwise $\mathbb{C}_i = false$. $\mathbb{T}_i$, $\mathbb{W}_i$ and $\mathbb{C}_i$ will be exploited in the next section to encode the scheduling targets in Cheng et al.'s work [20].

To make it easier to represent certain time points for fragment f_a^i, the following four symbols are defined:

(1) ES_a^i denotes the earliest start execution time of f_a^i, which indicates that any fragment f_a^i should start at ES_a^i or after ES_a^i, where $ES_a^i = r_i + \sum_{b=1}^{a-1} c_b^i$.

(2) LS_a^i denotes the latest start execution time of f_a^i, if $s_a^i > LS_a^i$, then the task τ_i will lose the meaning of accomplishment. $LS_a^i = d_i - \sum_{b=a}^{q_i} c_b^i$.

(3) EC_a^i denotes the earliest completion time of f_a^i, and obviously $EC_a^i = r_i + \sum_{b=1}^{a} c_b^i$, showing that any fragment f_a^i cannot end before EC_a^i.

(4) LC_a^i denotes the latest completion time of f_a^i, and obviously $LC_a^i = d_i - \sum_{b=a+1}^{q_i} c_b^i$. If fragment f_a^i cannot accomplish at LC_a^i or before LC_a^i, task τ_i will lose the meaning of accomplishment.

IV. An improved SMT encoding

In this section, we will elaborate on the optimization methods for Cheng et al. [20]. For the sake of clarity, the encoding is divided into two categories. One is the encoding of task attributes and the other is the encoding of scheduling targets. These encodings will be elaborated in the subsequent subsections.

A. Encoding task attributes

Cheng et al. [20] encoded four task attributes into SMT assertions, which are summarized as follows. In Z3 syntax, all these constraints are written in prefix expression, where the operator is put foremost, followed by a list of operands. Keyword "assert" indicates that the constraint should be satisfied with no exception. That is called a hard constraint.

(C1) $\forall \tau_i \in \mathcal{T}$, $(assert\ (>=\ s_1^i\ r_i))$

(C2) $\forall \tau_i \in \mathcal{T}, \forall f_a^i, f_b^j \in \tau_i, a < b$, $(assert (>= s_b^i (+ s_a^i c_a^i)))$

(C3) $\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall f_a^i \in \tau_i, \forall f_b^j \in \tau_j$,
$(assert\ (or\ (>=\ s_a^i\ (+\ s_b^j\ c_b^j))\ (>=\ s_b^j\ (+\ s_a^i\ c_a^i))))$

(C4) $\forall \tau_i, \tau_j \in \mathcal{T}, \tau_i \prec \tau_j$, $(assert\ (and\ (>=\ s_1^j\ (+ s_{q_i}^i\ c_{q_i}^i))$
$(\Rightarrow\ (>\ (+ s_{q_i}^i\ c_{q_i}^i)\ d_i)\ (>\ s_1^j\ d_j))))$

Constraint (C1) ensures that the first fragment of task τ_i starts not earlier than the request time of τ_i. (C2) ensures that the a^{th} fragment of task τ_i is executed prior to the b^{th} fragment of τ_i $(a < b)$. (C3) states either of the two fragments in different tasks executes prior to the other. (C4) ensures that when task τ_i depends on task τ_j, the last fragment of τ_j is executed prior to the first fragment of τ_i.

Although constraints (C1)-(C4) are complete and expressive enough to model the scheduling problems, there are redundant assertions, which may reduce the efficiency of problem solving.

First of all, the condition $i \neq j$ in (C3) leads to a large number of duplicated encodings. Strictly following the constraint (C3), we have to generate assertions like:

- $(assert\ (or\ (>=\ s_a^i\ (+\ s_b^j\ c_b^j))\ (>=\ s_b^j\ (+\ s_a^i\ s_a^i))))$
- $(assert\ (or\ (>=\ s_b^j\ (+\ s_a^i\ s_a^i))\ (>=\ s_a^i\ (+\ s_b^j\ c_b^j))))$

Obviously, these two assertions are equivalent due to the commutative law of "or" operations. In the following discussion of encoding redundancy, we replace $i \neq j$ with $i < j$ to eliminate such trivial duplication.

Example 1. Given a task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, tasks attributes are specified in a four-tuple, i.e., $\tau_i = (r_i, c_i, d_i, w_i)$, shown as follows:

- $\tau_1 = (0, 1, 2, 1)$
- $\tau_2 = (1, 1, 4, 1)$
- $\tau_3 = (2, 1, 4, 1)$

Assume that τ_2 depends on the computed results of τ_1, denoted by $\tau_1 \prec \tau_2$. For convenience, each task consists of only one fragment.

According to (C3), three assertions should be generated:

(1) $(assert\ (or\ (>=\ s_1^1\ (+\ s_1^2\ 1))\ (>=\ s_1^2\ (+\ s_1^1\ 1))))$

(2) $(assert\ (or\ (>=\ s_1^1\ (+\ s_1^3\ 1))\ (>=\ s_1^3\ (+\ s_1^1\ 1))))$

(3) $(assert\ (or\ (>=\ s_1^2\ (+\ s_1^3\ 1))\ (>=\ s_1^3\ (+\ s_1^2\ 1))))$

First, let us consider (1), which indicates f_1^1 starts after f_1^2 or f_1^2 after f_1^1. Since we have the constraint $\tau_1 \prec \tau_2$, explicitly forcing that τ_2 starts after the completion of τ_1 by (C4), assertion (1) makes no sense. In addition, constraint (2), which states that f_1^1 starts after f_1^3 or f_1^3 after f_1^1, is also redundant. According to the attributes of τ_1 and τ_3, it is clear that the deadline of τ_1 is equal to the request time of τ_3, both at time point 2. Consequently, there is no overlap in execution time period, and τ_3 starts after the execution of τ_1. Thus, explicitly specifying the execution order between these two tasks is unnecessary. Therefore, both (1) and (2) are redundant.

In order to eliminate the aforementioned redundancies of Cheng et al. [20], additional conditions are added to (C3), making it a new constraint ($\underline{C3}$) as follow.

($\underline{C3}$) $\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall f_a^i \in \tau_i, \forall f_b^j \in \tau_j$
$\tau_i \not\prec \tau_j, \tau_j \not\prec \tau_i, ES_b^j < LC_a^i, ES_a^i < LC_b^j$
$(assert\ (or\ (>=\ s_a^i\ (+\ s_b^j\ c_b^j))\ (>=\ s_b^j\ (+\ s_a^i\ c_a^i))))$,
where $i < j$

In constraint ($\underline{C3}$), conditions $\tau_i \not\prec \tau_j$ and $\tau_j \not\prec \tau_i$ eliminate the situation in which τ_i and τ_j have dependency relations, which has already captured by (C4). Conditions $ES_b^j < LC_a^i$ and $ES_a^i < LC_b^j$ guarantee that the execution time periods of f_a^i and f_b^j overlap. Only in this situation do we need to specify the execution orders of two fragments. After example 1 is re-encoded by using ($\underline{C3}$), only assertion (3) is retained. For convenience, ($\underline{C*}$) represents the conjuncted assertions in (C1), (C2), ($\underline{C3}$), and (C4).

B. Encoding scheduling targets

In ideal circumstances, the system scheduling goal is to complete each scheduled task before its deadline. When overload occurs, the goal is to maximize the number of tasks completed by the deadlines or maximize the sum of weights of tasks completed before their deadlines. These two targets are respectively formalized by the method of Cheng et al. [20] as the following assertions:

(T1) $Max(|\mathbb{T}|)$ where $\mathbb{T} \subseteq \mathcal{T}$, satisfying: $\forall \tau_i \in \mathcal{T}$,
$(assert\ (ite\ (<=\ (+\ s_{q_i}^i\ c_{q_i}^i)\ d_i))\ (=\ \mathbb{T}_i\ 1)\ (=\ \mathbb{T}_i\ 0))$

(T2) $Max(|\mathbb{W}|)$ where $\mathbb{W} \subseteq \mathcal{W}$, satisfying $\forall \tau_i \in \mathcal{T}$,
$(assert\ (ite\ (<=\ (+\ s_{q_i}^i\ c_{q_i}^i)\ d_i))\ (=\ \mathbb{W}_i\ w_i)\ (=\ \mathbb{W}_i\ 0))$

where the "$ite\ (c\ e_1\ e_2)$" is an "if-then-else" expression denoting e_1 if c is true and e_2 otherwise. For instance, $ite\ (>\ x\ y)\ (=\ x\ 0)\ (=\ y\ 0)$ means that if $x > y$, then $x = 0$, otherwise, $y = 0$.

In (T1), $|\mathbb{T}|$ represents the total number of tasks that can be completed before their deadlines. $|\mathbb{T}| = \sum_{i=1}^{|\mathcal{T}|} \mathbb{T}_i$, where $\mathbb{T}_i$ is an integer variable, representing the completion status with respect to τ_i. If τ_i is completed before its deadline, $\mathbb{T}_i$ is assigned with a bonus value of 1, otherwise, $\mathbb{T}_i$ equals 0. Clearly, $|\mathbb{T}|$ is in the interval of $[0,|\mathcal{T}|]$. To find the maximum $|\mathbb{T}|$, Cheng et al. [20] conducted binary search through $[0,|\mathcal{T}|]$ until a critical value of $|\mathbb{T}|$ is determined, with which the Z3 solver returns a non-empty model and an empty model with $|\mathbb{T}| + 1$.

In (T2), $|\mathcal{W}|$ represents the sum of weights of all tasks in $\mathcal{T}$, and $|\mathbb{W}|$ represents the total weight of tasks that are completed before their deadlines. $|\mathbb{T}| = \sum_{i=1}^{|\mathcal{W}|} \mathbb{W}_i$, where $\mathbb{W}_i$ is the weight obtained by τ_i. $\mathbb{W}_i$ is equal to w_i if τ_i is completed before the deadline, and 0 otherwise.

Obviously, $|\mathbb{W}|$ is in interval [0,$|\mathcal{W}|$]. Similar with (T1), to find out the optimum $|\mathbb{W}|$, Cheng et al. [20] conducted binary search to determine the maximum value of $|\mathbb{W}|$, with which the Z3 solver returns a non-empty model and an empty model with $|\mathbb{W}|+1$.

Although assertions in (T1) and (T2) guarantee a Z3 solver can output the correct scheduling result, and binary search, to some extent, facilitates the search of the optimal solution, it should be noted that the successive calls of a Z3 solver are likely to increase the overall computation time. Recently, motivated by the tremendous progress of SMT solvers, we take great advantage of the optimized version of Z3 and eliminate the cycle calls of the solver. In this section, assertions (T1) and (T2) are updated in two different ways, which are called METHOD1 and METHOD2 in the following. Both METHOD1 and METHOD2 eliminate successive Z3 calls.

To uptate the scheduling targets in Cheng et al.'s work [20], we propose compromised assertions $(T1')$ and $(T2')$ in METHOD1, which are listed as follows:

$(T1')$ $\forall \tau_i \in \mathcal{T}$, $(assert-soft\ \ \mathbb{C}_i)$
$(assert\ (ite\ (<=\ \ (+\ s^i_{q_i}\ c^i_{q_i})\ d_i))\ (=\ \ \mathbb{C}_i\ true)\ (=\ \mathbb{C}_i\ false))$

$(T2')$ $\forall \tau_i \in \mathcal{T}$, $(assert-soft\ \ \mathbb{C}_i : weight\ w_i)$
$(assert\ (ite\ (<=\ \ (+\ s^i_{q_i}\ c^i_{q_i})\ d_i))\ (=\ \ \mathbb{C}_i\ true)\ (=\ \mathbb{C}_i\ false))$

where $\mathbb{C}_i$ corresponds to an intermediate variable. The keyword "assert-soft" will make $\mathbb{C}_i$ equal to true as many as possible, and the value of $\mathbb{C}_i$ is controlled by the condition after the keyword "ite". If τ_i is completed before its deadline, $\mathbb{C}_i$ is true, otherwise, $\mathbb{C}_i$ equals to false. Under the condition that all the constraints in $(\underline{C*})$ are satisfied, the Z3 solver would maximize the number of completed tasks of satisfied assertions in $(T1')$ with the combination of $(\underline{C*})$ and $(T1')$. Similarly, when all constraints in $(\underline{C*})$ are satisfied, Z3 solver would maximize the sum of weights of satisfied assertions in $(T2')$ with the combination of $(\underline{C*})$ and $(T2')$. The keyword "weight" in $(T2')$ specifies the obtained value when the soft assertion is satisfied. Weight w_i is gained by the solver when the assertion is satisfied, otherwise, the gain is 0.

Although assertions $(T1')$ and $(T2')$ modified by METHOD1 seem to be perfect, it should be noted that METHOD1 uses intermediate variables, which may increase the processing steps of Z3. In what we follows, we present another updated assertions, which are summarized in METHOD2, listed as follows:

$(\underline{T1})$ $\forall \tau_i \in \mathcal{T}$, $(assert-soft\ (<=\ (+\ s^i_{q_i}\ c^i_{q_i})\ d_i))$

$(\underline{T2})$ $\forall \tau_i \in \mathcal{T}$, $(assert{-}soft(<= (+s^i_{q_i} c^i_{q_i}) d_i) : weight\ w_i)$

where "assert-soft" is a keyword provided by the optimized version of Z3. All constraints defined after this keyword are declared as soft constraints, which should be satisfied to the largest extent. $(\underline{T1})$ states that $\forall \tau_i \in \mathcal{T}$, that τ_i is completed before its deadline is a soft constraint. Combined with $(\underline{C*})$ and (T1), the Z3 solver would maximize the satisfied assertions in $(\underline{T1})$, provided that all the constraints in $(\underline{C*})$ are met.

Similarly, combined with $(\underline{C*})$ and (T2), the Z3 solver would maximize the sum of weights of satisfied assertions in $(\underline{T2})$, provided that all the constraints in $(\underline{C*})$ are met. The keyword "weight" in $(\underline{T2})$ specifies the obtained value when the soft assertion is satisfied. In this case, weight w_i is gained by the solver, otherwise, the gain is 0. By using the keywords "assert-soft" and "weight", the scheduling target can be achieved without successive calls of the Z3 solver. To sum up, we have updated SMT formulation with Z3 syntax on the basis of Cheng et al.'s work [20] by METHOD1 and METHOD2. The modifications are two-folded. First, more conditions are added to encode the task attributes, making the encoding more concise. Second, we have made full use of the keywords "assert-soft" and "weight" in Z3 syntex to eliminate successive calls of the Z3 solver. With the updated formulation, the optimal scheduling algorithm based on SMT becomes more compact and efficient.

TABLE II
Critical time instant of each fragment in four tasks

Task	Fragment	ES	LS	EC	LC
τ_1	f_1^1	1	4	2	5
	f_2^1	2	5	3	6
	f_3^1	3	6	4	7
τ_2	f_1^2	0	0	5	5
τ_3	f_1^3	0	2	4	6
τ_4	f_1^4	3	7	4	8

C. A pedagogical example

How the SMT formulation works is described considering a simple scheduling problem. A set of real-time tasks $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ are assumed. The request time instant r_i, required execution time c_i, deadline d_i, and weight w_i of task τ_i are expressed in a four tuple (r_i, c_i, d_i, w_i). τ_1 has three fragments, and the execution time of each fragment is 1. The rest tasks τ_2, τ_3, and τ_4 have only one fragment, and the execution time of every fragment is 5, 4, and 1, respectively. In addition, τ_4 relies on τ_2, indicating that τ_4 is executed after the end of τ_2.

- $\tau_1 = (1, 3, 7, 1), q_1 = 3$
- $\tau_2 = (0, 5, 5, 2), q_2 = 1$
- $\tau_3 = (0, 4, 6, 1), q_3 = 1$
- $\tau_4 = (3, 1, 8, 3), q_4 = 1$
- $\tau_2 \prec \tau_4$

In this scheduling problem, the critical time instant of each fragment in the four tasks is summarized in Table II.

The SMT formulations applied to the scheduling problem with the Z3 solver are shown in Figure 1, 2, and 4. Figure 1 shows the encoded task attributes, and the encoded scheduling targets are shown in Figure 2 and Figure 4 with METHOD1 and METHOD2, respectively. Constraint (C1) states that a task is execute after it requests to run, which means that the start time of the first fragment of a task should be longer than or equal to its ES. τ_1 consists of three fragments f_1^1, f_2^1, and f_3^1. The start time of f_1^1 should be longer than or equal to ES_1^1. Constraint (C2) ensures that the series of fragments in a task should be executed sequentially. For example, τ_1 consists of three fragments f_1^1, f_2^1, and f_3^1, $s_2^1 \geq s_1^1 + c_1^1$,

(C1) $(assert\,(>=\ s_1^1\,1))$
$(assert\,(>=\ s_1^2\,0))$
$(assert\,(>=\ s_1^3\,0))$
$(assert\,(>=\ s_1^4\,3))$
(C2) $(assert\,(>=\ s_2^1(+\,s_1^1\,1)))$
$(assert\,(>=\ s_3^1(+\,s_1^1\,1)))$
$(assert\,(>=\ s_3^1(+\,s_2^1\,1)))$
($\underline{C3}$) $(assert\,(or\,(>=\ s_1^1(+\,s_1^2\,5))(>=\ s_1^2(+\,s_1^1\,1))))$
$(assert\,(or\,(>=\ s_2^1(+\,s_1^2\,5))(>=\ s_1^2(+\,s_2^1\,1))))$
$(assert\,(or\,(>=\ s_3^1(+\,s_1^2\,5))(>=\ s_1^2(+\,s_3^1\,1))))$
$(assert\,(or\,(>=\ s_1^1(+\,s_1^3\,4))(>=\ s_1^3\,(+\,s_1^1\,1))))$
$(assert\,(or\,(>=\ s_2^1(+\,s_1^3\,4))(>=\ s_1^3\,(+\,s_2^1\,1))))$
$(assert\,(or\,(>=\ s_3^1(+\,s_1^3\,4))(>=\ s_1^3\,(+\,s_3^1\,1))))$
$(assert\,(or\,(>=\ s_1^1(+\,s_1^4\,1))(>=\ s_1^4\,(+\,s_1^1\,1))))$
$(assert\,(or\,(>=\ s_2^1(+\,s_1^4\,1))(>=\ s_1^4\,(+\,s_2^1\,1))))$
$(assert\,(or\,(>=\ s_3^1(+\,s_1^4\,1))(>=\ s_1^4\,(+\,s_3^1\,1))))$
$(assert\,(or\,(>=\ s_1^2(+\,s_1^3\,4))(>=\ s_1^3\,(+\,s_1^2\,5))))$
$(assert\,(or\,(>=\ s_1^3(+\,s_1^4\,1))(>=\ s_1^4\,(+\,s_1^3\,4))))$
(C4) $(assert\,(and\,(>=\ s_1^4(+\,s_1^2\,5))(\Rightarrow\,(>\,(+\,s_1^2\,5)\,5)\,(>\,s_1^4\,8))))$

Fig. 1. Modified Z3-coded SMT tasks attribute constraints for the examplified problem

$(T1')$
$(assert-soft\ \ \mathbb{C}_1)$
$(assert-soft\ \ \mathbb{C}_2)$
$(assert-soft\ \ \mathbb{C}_3)$
$(assert-soft\ \ \mathbb{C}_4)$
$(assert\,(ite\,(<=\,(+\,s_3^1\,1)7)(=\,\mathbb{C}_1\,true)(=\,\mathbb{C}_1\,false)))$
$(assert\,(ite\,(<=\,(+\,s_1^2\,5)5)(=\,\mathbb{C}_2\,true)(=\,\mathbb{C}_2\,false)))$
$(assert\,(ite\,(<=\,(+\,s_1^3\,4)6)(=\,\mathbb{C}_3\,true)(=\,\mathbb{C}_3\,false)))$
$(assert\,(ite\,(<=\,(+\,s_1^4\,1)8)(=\,\mathbb{C}_4\,true)(=\,\mathbb{C}_4\,false)))$

$(T2')$
$(assert-soft\ \ \mathbb{C}_1 : weight\,1)$
$(assert-soft\ \ \mathbb{C}_2 : weight\,2)$
$(assert-soft\ \ \mathbb{C}_3 : weight\,1)$
$(assert-soft\ \ \mathbb{C}_4 : weight\,3)$
$(assert\,(ite\,(<=\,(+\,s_3^1\,1)7)(=\,\mathbb{C}_1\,true)(=\,\mathbb{C}_1\,false)))$
$(assert\,(ite\,(<=\,(+\,s_1^2\,5)5)(=\,\mathbb{C}_2\,true)(=\,\mathbb{C}_2\,false)))$
$(assert\,(ite\,(<=\,(+\,s_1^3\,4)6)(=\,\mathbb{C}_3\,true)(=\,\mathbb{C}_3\,false)))$
$(assert\,(ite\,(<=\,(+\,s_1^4\,1)8)(=\,\mathbb{C}_4\,true)(=\,\mathbb{C}_4\,false)))$

Fig. 2. Modified Z3-coded SMT target constraints with METHOD1 for the examplified problem

($\underline{T1}$) $(assert-soft\,(<=\ (+\,s_3^1\,1)\,7))$
$(assert-soft\,(<=\ (+\,s_1^2\,5)\,5))$
$(assert-soft\,(<=\ (+\,s_1^3\,4)\,6))$
$(assert-soft\,(<=\ (+\,s_1^4\,1)\,8))$

($\underline{T2}$) $(assert-soft\,(<=\ (+\,s_3^1\,1)\,7):\ weight\,1)$
$(assert-soft\,(<=\ (+\,s_1^2\,5)\,5):\ weight\,2)$
$(assert-soft\,(<=\ (+\,s_1^3\,4)\,6):\ weight\,1)$
$(assert-soft\,(<=\ (+\,s_1^4\,1)\,8):\ weight\,3)$

Fig. 3. Modified Z3-coded SMT target constraints with METHOD2 for the examplified problem

$s_3^1 \geq s_1^1 + c_1^1$, and $s_3^1 \geq s_2^1 + c_2^1$. Constraint ($\underline{C3}$) states that if two independent fragments overlap in their execution period, the execution order should be explicitly specified, so that a processor execute only one fragment at a time. Constraint (C4) deals with the situation in which tasks have dependency relation with each other. Take τ_2 and τ_4 as an example. Since $\tau_2 \prec \tau_4$, τ_4 can only run after τ_2 finishes.

As for the scheduling targets, Z3 satisfies the assertions in constraints $(T1')$ and ($\underline{T1}$) to the largest extent with the keyword "assert-soft" and satisfies the constraint $(T2')$ and ($\underline{T2}$) with keywords "weight" and "assert-soft".

If the scheduling target is to maximize the number of the completed tasks, we will conjunct all task attribute constraints ($\underline{C*}$) and target constraints $(T1')$ or ($\underline{T1}$). If some tasks are more important than others and are scheduled with higher priority, we will conjunct all task constraints ($\underline{C*}$) and target constraints $(T2')$ or ($\underline{T2}$). All these constraints are combined together and form a SMT problem, which is called a SMT model. After this SMT model is imported into Z3 solver, we can get a scheduling table by the returned non-empty model.

Because METHOD1 uses the intermediate variable $\mathbb{C}_i$, it increases the in and out of the stack's times of Z3, and extra time is needed to generate the intermediate results. As a result, METHOD2 is more efficient than METHOD1. In our experiments, it is also proved that METHOD2 is faster than METHOD1.

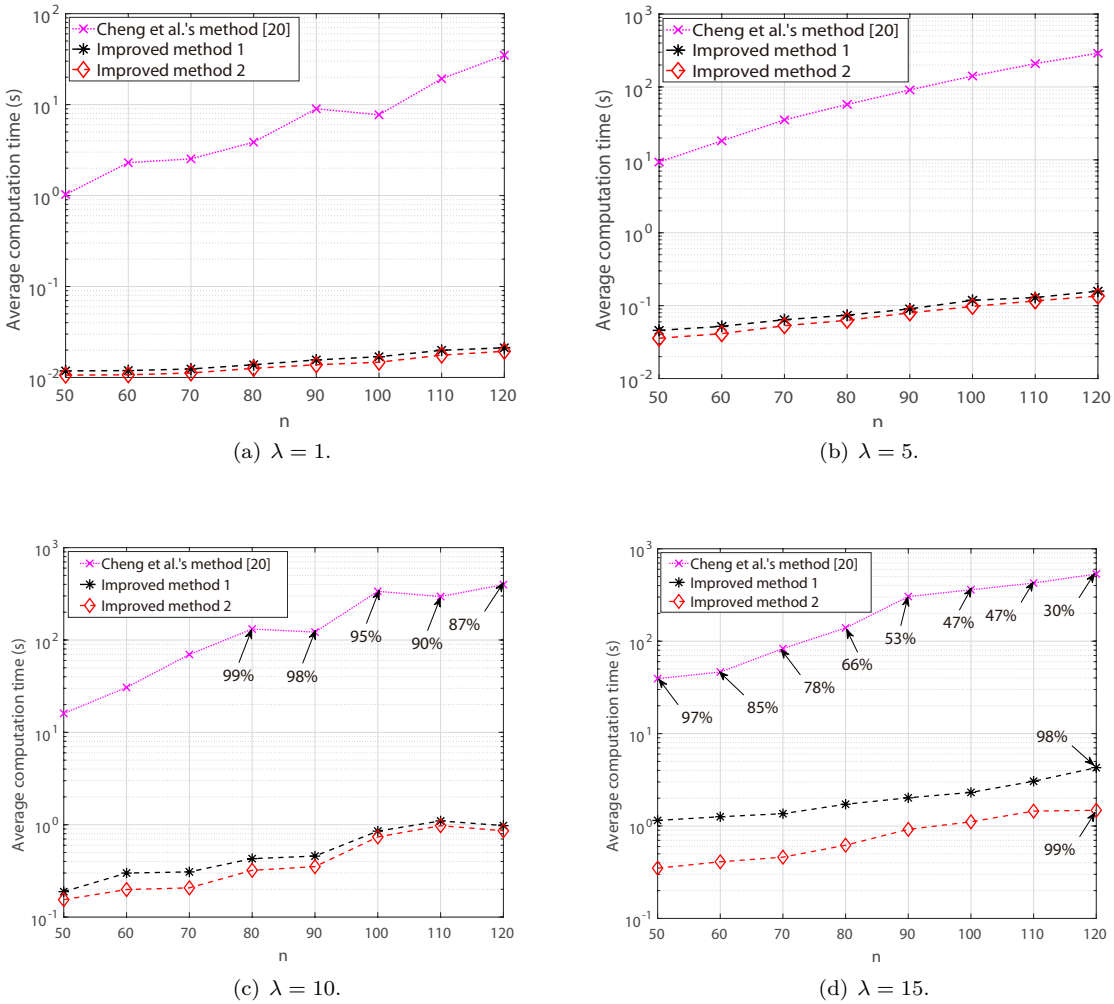
TABLE III
Parameter setting in experiments

Parameter	Description	Value Setting
λ	Arriving rate	$\{1, 5, 10\}$
n	The total number of tasks in $\mathcal{T}$	[50,120]
sf_i	Slack factor of τ_i	$[1, 4]$
c_i	Execution time of τ_i	$[1, 13]$
q_i	The number of fragments in τ_i	$[1, 3]$
d_i	Deadline of τ_i	$d_i = r_i + sf_i * c_i$
w_i	Weight of τ_i	$[1, n]$

V. Experiments

A. Experimental Settings

We follow the method of creating scheduling problems in the work [20], which is summarized as follows. The tasks are created according to uniform distribution with arriving rate λ, which represents the number of tasks that arrive per 100 time units. It is obvious that the larger the λ is, the more serious the overload of the system. In our experiments, λ is assigned with 1, 5, and 10 to represent various degrees of system overload. For each λ, a set of real-time tasks $\mathcal{T}$ are generated, where the total number of tasks in $\mathcal{T}$, denoted by n, ranges from 50 to 120. For each task τ_i, the execution time c_i ranges from 1 to 13, and the number of fragments in τ_i, denoted by q_i, ranges from 1 to 3. The value of deadline d_i is calculated by the formula $d_i = r_i + sf_i * c_i$, where sf_i is the slack factor



(a) $\lambda = 1$. (b) $\lambda = 5$.

(c) $\lambda = 10$. (d) $\lambda = 15$.

Fig. 4. Average computation time of three SMT encodings with scheduling target: maximizing the number of tasks completed before their deadlines.

that indicates the tightness of task deadline. We suppose that sf_i ranges from 1 to 4 for each task τ_i. Each task τ_i is associated with an integer weight w_i to reflect its importance. The weight w_i is uniformly chosen between 1 and the number of tasks at random. For each fixed λ and n, 100 problem instances are generated, and the number of task dependency relation is randomly assigned to 10% of n. For convenience, the parameter settings are listed in Table III.

In the following experiments, we carried out tests on Linux virtual machine with version Ubuntu 16.04 on VMware Workstation 14, and the machine is i7-6700HQ at 2.60GHz with 8GB Intel(R) Core(TM). Z3 was used as the core SMT solver. All methods were run on the same set of test cases, and their results are directly comparable.

B. Comparisons of Three SMT Encodings

To decrease unnecessary experiments and initially determine the optimal scheduling method, the following three methods are evaluated:

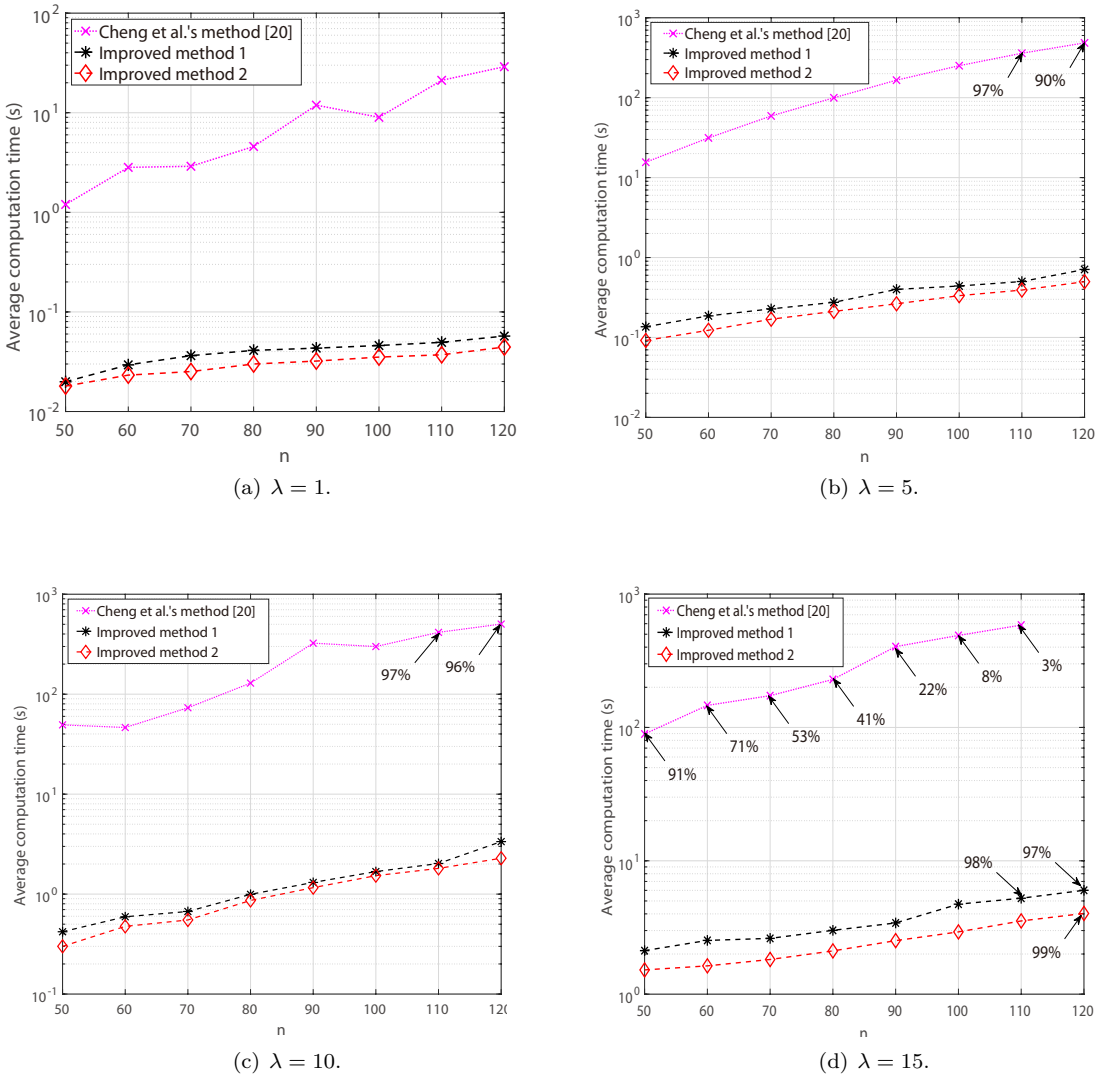
(1) Cheng et al.'s method, which is the primitive method presented in [20]. The method has a lot of redundant encodings and successive SMT calls.

(2) Improved method 1, which combines ($\underline{C*}$) with METHOD1 by replacing (T1) and (T2) with ($T1'$) and ($T2'$).

(3) Improved method 2, which combines ($\underline{C*}$) with METHOD2 by replacing (T1) and (T2) with ($\underline{T1}$) and ($\underline{T2}$).

Figures 4 and 5 depict the average computation time of these three encoding. The scheduling target is to maximize the number of tasks completed before the deadlines (shown in Figure 4) and maximizing the sum of weights of completed tasks (shown in Figure 5). Each data point is the average computation time of the solved problem instances. The number with an arrow shown in the figures means the percent age of the instances successfully solved within the time limit by Z3, and it is omitted if Z3 managed to slove all the 100 instances. When the percentage of the solved instances drops to zero, the corresponding curve is omitted as the average computation time becomes unpredictable.

As shown in Figures 4 and 5, no matter how λ and n change, both improved method 1 and improved method 2 are far better than Cheng et. al's method [20], and improved method 2 is invariably superior to the other two methods. When λ equals to 1, 5 and 10, the overload of the system is not so serious, and the process of solving



(a) $\lambda = 1$. (b) $\lambda = 5$.

(c) $\lambda = 10$. (d) $\lambda = 15$.

Fig. 5. Average computation time of three SMT encodings with scheduling target: maximizing the number of tasks completed before their deadlines.

the optimal scheduling results is not so complicated. As a result, the curve representing improved method 1 is quite close to the curve representing improved method 2, but the former is still above the latter. When λ increases to 15, the system overload situation becomes more serious. Compared with improved method 2, the increased number of solving steps in improved method 1 influences the final computation time more significantly. Thus, the curves of improved method 1 and improved method 2 show more obvious intervals, and improved method 2 is more efficient.

Specifically, as can be seen in Figure 4, when $\lambda = 15$, improved method 2 took around 1.5 seconds on average to solve instances with $n = 120$, while improved method 1 took around 4 seconds, which is more than twice as much as improved method 2. Meanwhile, the number of tasks which are not solved by improved method 1 within the time limit is twice that of improved method 2, and the original Cheng et al.'s method [20] consumes more than 500 seconds for 30% instances, with the rest 70% instances unsolved within the time limit.

Similar results can be found in Figure 5. When $\lambda = 15$ and $n = 110$, improved method 2 can tackle all instances in the limited time, while two instances are not solved by improved method 1. Cheng et. al's method [20] can only solve three instances, which makes the system almost unable to run.

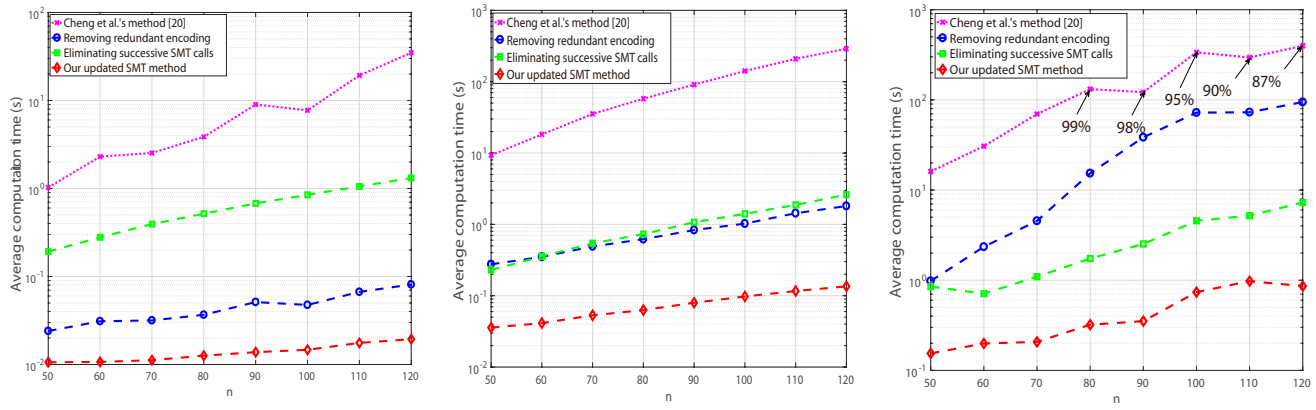
According to the results, compared with Cheng et al.'s method [20], improved method 1 and improved method 2 can greatly improve the solving efficiency. No matter how λ and n change, improved method 2 always outperforms improved method 1.

C. Extensive Experiments on Improved Method 2

According to Subsection V.B, it can be found that the improved method 2 have better performance than other methods. In this subsection, we further investigate how this method improves the problem solving efficiency by evaluating the following four methods:

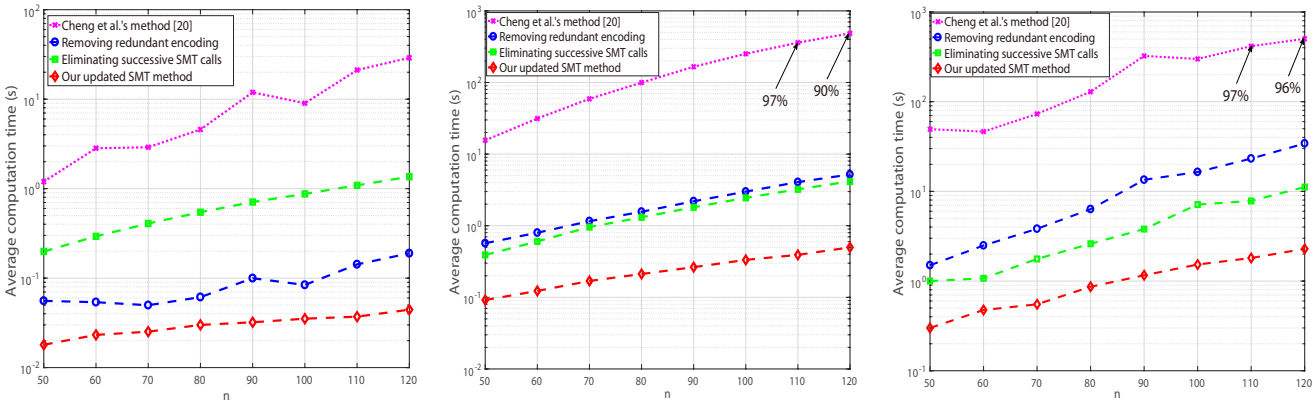
(1) Cheng et al.'s method. This is the original SMT-based formulation presented in [20], and it is consistent with the above experiment.

(2) Removing redundant encoding, which is updated from [20] by replacing constraint (C3) with ($\underline{C3}$).



(a) $\lambda = 1$. (b) $\lambda = 5$. (c) $\lambda = 10$.

Fig. 6. Average computation time of four SMT encodings with scheduling target: maximizing the number of tasks completed before their deadlines.



(a) $\lambda = 1$. (b) $\lambda = 5$. (c) $\lambda = 10$.

Fig. 7. Average computation time of four SMT encodings with scheduling target: maximizing the sum of weights of tasks completed before their deadlines.

This method removes redundant encoding, but successive calls of the Z3 solver are needed to find out the optimal solution.

(3) Eliminating successive SMT calls, which is updated from [20] by replacing constraints (T1) and (T2) with ($\underline{T1}$) and ($\underline{T2}$) through METHOD1. This method eliminates successive SMT calls, but retains redundant encoding.

(4) Improved method 2. This is our fasted improved SMT formulation which combines both updates in (2) and (3) from the above-mentioned experiment.

Figures 6 and 7 show the average computation time of these four encodings to generate the optimal scheduling, and the scheduling objectives are the same as those depicted in Figures 4 and 5 respectively. The meanings of each data point in Figures 6 and 7 are consistent with those in Figures 4 and 5, and for convenience, we will not elaborate here.

Figures 6 and 7 demonstrate that our updated SMT-based method is invariably superior to the other three methods, no matter how λ and n change. To be specific, as can be seen in Figure 6, when $\lambda = 10$, our updated SMT method took less than one second on average to solve instances with $n = 120$, and the original Cheng et al.'s method [20] consumed around 400 seconds for 87% instances, with the rest 13% instances unsolved within the time limit. In the middle are the methods of removing redundant encoding and eliminating successive SMT calls. The comparison results of these two methods differ as the overload degree changes, which is adjusted by parameter λ. When $\lambda = 1$, the curve with rectangles is always above the curve with hollow circles, indicating that eliminating redundant encoding at this point can reduce the computation time more significantly than eliminating successive SMT calls. By contrast, when $\lambda = 5$, the performance of these two methods becomes comparable. When $\lambda = 10$, eliminating successive SMT calls is dominant in reducing the overall computation time. The comparison results are briefly explained as follows. When λ is small, the overload degree is not severe and it is easy to run the SMT solver once to check whether the currently preset number of tasks would lead to an empty model. In this situation, the main factor that lowers the solving efficiency is the redundant encoding. After redundant encoding is eliminated, the enhanced efficiency is great enough to compensate the additional time consumed on successive SMT calls. In comparison, when λ is large, the increase of the overload degree makes

TABLE IV
The average number of assertions generated by Cheng et al.'s method and Removing redundant encoding

	$\lambda = 1$		$\lambda = 5$		$\lambda = 10$	
n	Cheng et al.'s method	Removing redundant encoding	Cheng et al.'s method	Removing redundant encoding	Cheng et al.'s method	Removing redundant encoding
50	10228.21	401.14	9964.30	649.86	10079.96	958.84
60	14738.08	474.73	14478.28	778.93	14703.89	1168.81
70	19841.32	557.41	19976.22	928.71	19634.30	1435.00
80	26238.10	647.14	25845.15	1093.91	25885.74	1663.07
90	32700.64	724.21	32695.55	1222.09	32555.71	1852.42
100	39781.03	785.88	40288.00	1412.56	39996.73	1959.84
110	48376.08	872.45	48830.94	1584.31	46425.50	2087.53
120	58197.69	952.72	58045.52	1746.19	56082.78	2442.00

the problem so difficult that running the SMT solver once consumes a large amount of time, let alone repeated calls of the solver. Thus, successive calls of the SMT solver becomes the overwhelming influencing factor of efficiency.

Similar results can be found in Figure 7, showing that when λ is small, removing redundant encoding plays a dominant role in reducing the computation time. With the increase of λ, eliminating successive SMT calls gradually becomes the noticeable factor to enhance the efficiency.

According to the experiment results, compared with Cheng et al.'s method [20], improved method 2, which eliminates redundant encoding and successive SMT calls, can improve the solving efficiency by more than two orders of magnitude.

In what follows, the number of assertions is compared quantitatively. Considering that a great multitude of assertions are derived from the redundant encoding, we only investigate the percentage of assertions that have been reduced by removing redundant encoding.

Figure 8 shows the evaluation results for scheduling target that maximizes the number of the completed tasks[1]. Evidently, after removing redundant encoding, the number of assertions has been reduced by more than 90%, and the reduced proportion increases continuously with the increase of the total number of tasks. On the other hand, we should notice that such proportional reduction is trending downward as λ rises. Particularly, given n is fixed at 120, when $\lambda = 1$, the percentage of the reduced assertions is over 98%, which is less than 96% when $\lambda = 10$. The reason for the downward trend is explained as follows. Smaller λ indicates that fewer tasks arrive at the system per 100 time units. Consequently, fewer pairs of tasks have overlap in their execution period. According to the updated condition ($\underline{C3}$), fewer assertions are required. As λ gets larger, more tasks arrive per 100 time units, and more tasks tend to overlap in their execution time. Therefore, their execution order should be explicitly specified by ($\underline{C3}$) to prevent two tasks from being executed simultaneously. As a result, the number of assertions generated by ($\underline{C3}$) increases as λ goes up. By contrast, (C3), which is adopted by Cheng et al. [20], generates a constant number of assertions, no matter whether tasks have overlap in their execution period. In other words, the number of assertions generated by Cheng et al's method is not dependent on the value of λ. As a result, the percentage of the reduced assertions decreases as λ increases.

[1]Similar results could be obtained from the target that maximizes the sum of weights of completed tasks. For conciseness, we only exhibit one figure.

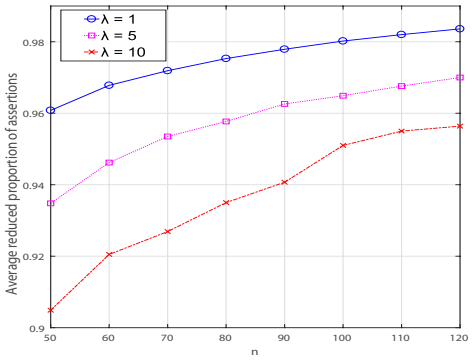


Fig. 8. Average reduced proportion of assertions caused by the step of removing redundant encoding with scheduling target: maximizing the number of tasks completed before their deadlines.

To verify our analysis, we further investigated the number of assertions generated by Cheng et al. [20] and the method of removing redundant encoding. Statistical results are exhibited in Table IV. Given a fixed n, the average number of assertions generated by Cheng et al's method is basically unchanged with the increase of λ, while that by the method of removing redundant encoding increases gradually. For example, given $n = 100$, as λ increases from 1 to 10, the number of assertions in Cheng et al.'s method is kept constantly around $40,000$, while the value drops from 785.88 to 1959.84 after removing redundant encoding. As a result, the average reduced proportion decreases from 98.02% to 95.10%. This demonstrates the exhibition of Figure 5, showing that the curve with smaller λ is above that with larger λ. Furthermore, we explored the number of assertions, and found that after removing redundant encoding, the number of assertions was reduced by more than 90%. This further demonstrates the great advantage of the updated SMT formalization.

VI. Conclusion

Applying SMT to scheduling problems in real-time systems has been proved to be efficient in previous research [20]. In the present study, the SMT-based method of Cheng et al. [20] was improved by removing redundant encoding and successive calls of the SMT solver. Experiment results demonstrate that efficiency of the improved method significantly outperforms the previous works by more than two orders of magnitude, no matter how the number of tasks and the degree of overload change.

References

[1] W. L. Al-Yaseen, Z. A. Othman, and M. Z. A. Nazri, "Real-time intrusion detection system using multi-agent system," IAENG International Journal of Computer Science, vol. 43, no. 1, pp. 80–90, 2016.

[2] S. Choi and C. Poon, "An rfid-based anti-counterfeiting system." IAENG International Journal of Computer Science, vol. 35, no. 1, pp. 80–91, 2008.

[3] M. K. Gardner and J. W. S. Liu, "Performance of algorithms for scheduling real-time systems with overrun and overload," in Euromicro Conference on Real-time Systems, 1999.

[4] A. Burns, "Scheduling hard real-time systems: a review," Software Engineering Journal, vol. 6, no. 3, pp. 116–128, 1991.

[5] S. K. Baruah and J. R. Haritsa, "Scheduling for overload in real-time systems," Computers IEEE Transactions on, vol. 46, no. 9, pp. 1034–1039, 1997.

[6] S. Baruah, J. Haritsa, and N. Sharma, "On-line scheduling to maximize task completions," in Real-time Systems Symposium, 1994.

[7] C. Tres, L. B. Becker, and E. Nett, "Real-time tasks scheduling with value control to predict timing faults during overload," in 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07). IEEE, 2007, pp. 354–358.

[8] A. Marchand and M. Chetto, "Dynamic scheduling of periodic skippable tasks in an overloaded real-time system," in 2008 IEEE/ACS International Conference on Computer Systems and Applications. IEEE, 2008, pp. 456–464.

[9] Z. Cheng, H. Zhang, Y. Tan, and A. O. Lim, "Dpsc: A novel scheduling strategy for overloaded real-time systems," in 2014 IEEE 17th International Conference on Computational Science and Engineering. IEEE, 2014, pp. 1017–1023.

[10] ——, "Greedy scheduling with feedback control for overloaded real-time systems." in Ifip/ieee International Symposium on Integrated Network Management, 2015.

[11] K. Kuchcinski, "Constraints-driven scheduling and resource assignment," Acm Transactions on Design Automation of Electronic Systems, vol. 8, no. 3, pp. 355–383, 2003.

[12] W. Liu, Z. Gu, J. Xu, X. Wu, and Y. Ye, "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 8, pp. 1382–1389, 2010.

[13] J. M. Crawford and A. B. Baker, "Experimental results on the application of satisfiability algorithms to scheduling problems," in Twelfth Aaai National Conference on Artificial Intelligence, 1994.

[14] J. Franco and J. Martin, "A history of satisfiability." Handbook of satisfiability, vol. 185, pp. 3–74, 2009.

[15] S. Gubin, "Polynomial size asymmetric linear model for sat," in Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008. IEEE, 2008, pp. 62–66.

[16] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in Handbook of Model Checking. Springer, 2018, pp. 305–343.

[17] L. De Moura and N. Bjørner, "Satisfiability modulo theories: introduction and applications," Communications of the ACM, vol. 54, no. 9, pp. 69–77, 2011.

[18] C. Zhuo, H. Zhang, Y. Tan, and Y. Lim, "Scheduling overload for real-time systems using smt solver," in IEEE/ACIS International Conference on Software Engineering, 2016.

[19] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "Smt-based scheduling for multiprocessor real-time systems," in 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS). IEEE, 2016, pp. 1–7.

[20] ——, "Smt-based scheduling for overloaded real-time systems," IEICE TRANSACTIONS on Information and Systems, vol. 100, no. 5, pp. 1055–1066, 2017.

[21] V. Popov, "The approximate period problem," IAENG International Journal of Computer Science, vol. 36, no. 4, pp. 268–274, 2009.

[22] A. Gorbenko, M. Mornev, and V. Popov, "Planning a typical working day for indoor service robots," IAENG International Journal of Computer Science, vol. 38, no. 3, pp. 176–182, 2011.

[23] A. Gorbenko and V. Popov, "The c-fragment longest arc-preserving common subsequence problem," IAENG International Journal of Computer Science, vol. 39, no. 3, pp. 231–238, 2012.

[24] J. Blazewicz, J. K. Lenstra, and A. R. Kan, "Scheduling subject to resource constraints: classification and complexity," Discrete applied mathematics, vol. 5, no. 1, pp. 11–24, 1983.

[25] M. Bofill, J. Coll, J. Suy, and M. Villaret, "Solving the multi-mode resource-constrained project scheduling problem with smt," in 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, 2016, pp. 239–246.

[26] H. Takamatsu, H. Sato, S. Oyama, and M. Kurihara, "Automated test generation for object-oriented programs with multiple targets," IAENG International Journal of Computer Science, vol. 41, no. 3, pp. 198–203, 2014.

[27] L. D. Moura and N. Bjørner, "Z3: An efficient smt solver," 2008.

[28] L. Moura and N. Bjørner, "Satisfiability modulo theories: An appetizer." Lecture Notes in Computer Science, vol. 5902, pp. 23–36, 2009.

[29] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The mathsat5 smt solver," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2013, pp. 93–107.

[30] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νz-an optimizing smt solver," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2015, pp. 194–199.

[31] R. Sebastiani and P. Trentin, "Optimathsat: A tool for optimization modulo theories," in International conference on computer aided verification. Springer, 2015, pp. 447–454.

[32] S. F. Roselli, K. Bengtsson, and K. Åkesson, "Smt solvers for job-shop scheduling problems: Models comparison and performance evaluation," in 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE). IEEE, 2018, pp. 547–552.

[33] J. R. Haritsa, "On being optimistic about real-time constraints," in Acm Sigact-sigmod-sigart Symposium on Principles of Database Systems, 1990.